J Real-Time Image Proc (2012) 7:185–202
DOI 10.1007/s11554-011-0195-8

SPECIAL ISSUE

Task complexity analysis and QoS management for mapping dynamic video-processing tasks on a multi-core platform

A. H. R. Albers · P. H. N. de With

Received: 15 July 2009 / Accepted: 18 January 2011 / Published online: 26 February 2011

Abstract This paper addresses efficient mapping and reconfiguration of advanced video applications onto a general purpose multi-core platform. By accurately modeling the resource usage for an application, allocation of processing resources on the platform can be based on the actually needed resources instead of a worst-case approach, thereby improving Quality-of-Service (QoS). Here, we exploit a new and strongly upcoming class of dynamic video applications based on image and content analysis for resource management and control. Such applications are characterized by irregular computing behavior and memory usage. It is shown that with linear models and statistical techniques based on the Markov modeling, a rather good accuracy (94–97%) for predicting the resource usage can be obtained. This prediction accuracy is so good that it allows resource prediction at runtime, thereby leading to an actively controlled system management.

Keywords Video processing · Performance modeling · Multiprocessing · Object recognition · Stochastic approximation

A. H. R. Albers (✉)
Interventional X-ray, R&D, Philips Healthcare, P.O. Box 10.000, 5680 DA Best, The Netherlands
e-mail: R.Albers@philips.com

P. H. N. de With
Electrical Engineering, Signal Processing Systems, Eindhoven University of Technology, P.O. Box 513, LG 0.28, 5600 MB Eindhoven, The Netherlands
e-mail: P.H.N.de.With@tue.nl

1 Introduction

1.1 Preliminaries

The design of video signal processing systems is gradually entering a new phase, where instead of straightforward video processing for multimedia and quality improvements, the systems adapt the processing to the platform and content conditions by considering in detail the actual content that is being processed. This ranges from extracting simple features, such as color and texture, to an in-depth analysis of the video signal and its characteristic features. Examples of such applications are intelligent surveillance, face and behavior recognition, computer analysis of diseases in the medical domain, etc. In addition, computing platforms for executing these processing algorithms have been subject to continuous changes, but are now gradually converging towards multi-core processor architectures, where depending on cost and application constraints, some of the cores are application specific and others are fully programmable. At first glance, video processing is well suited for multi-core processing, given the possibilities for dividing the tasks and the amounts of data that are involved in this type of processing, but the mapping on the platform and the optimal distribution of tasks is a complex matter.

With respect to platforms, the trend to go to multi-core systems is fueled by the desire to further improve the computing power, while maintaining energy efficiency. These architectures are characterized by a set of parallel processing cores on a chip centered around a communication infrastructure, which is connected to a large off-chip memory. Chip connections and power consumption for drivers hamper the use of high-bandwidth buses, so that memory bandwidth cannot grow with the same speed as computation. Unfortunately, the mapping of complicated

video processing tasks is difficult on such parallel platforms, because compilers for automated mapping do not exist or perform poorly. The system designer has no other alternative than to analyze the applications with respect to their computing requirements and behavior and divide the processing tasks over the multiple cores on the system accordingly. This creates a demand for techniques to facilitate the design of such systems.

In competitive imaging markets, such as professional healthcare, surveillance or multimedia, computer platforms are subject to a restricted bill of material. To be cost-effective, it is required that the applications make efficient use of the available resources on the platform. The mapping and control of multiple applications onto a multi-core system is an unsolved case in the scientific literature. This is due to the large design space and the fact that operating systems have no understanding about the consequences of scheduling decisions to the individual applications in terms of throughput, latency or visual quality. It is only recently that the first publications on this topic have become available. In a paper from [5], an integration of hard/soft real-time tasks and best-effort jobs executed on a multiprocessor system is studied. The study involves computing only and does not address other resources. The optimal scheduling of tasks over a multiprocessor system is not yet found. If the processors operate independently of each other, then the solution of [6] can be applied on an individual processor basis. The work of Pastrnak et al. [30] introduces a hierarchical Quality-of-Service (QoS) protocol for monitoring the performance of streaming tasks. In this way, the multiprocessor system can be fully active and deliver the highest possible quality for the most prioritized computing tasks. As the validation of this framework was limited to streaming tasks only, it is not sure whether this can be used for more modern video processing such as content analysis and object-based processing.

This paper contributes to this problem by concentrating on the control of both the computing and the bandwidth between the processing cores for image analysis applications. For validation of our contribution, we will explore a case study for medical image analysis in the professional domain. Medical imaging is a fast growing imaging domain with challenges on fusion of sensor data and the automatic computer analysis of diseases. The application described in this paper is carefully chosen to serve as a general representative case for modern video processing, as the underlying processing tasks also appear in many other application domains or systems.

1.2 Scope and problem statement

The scope of this paper is to build a system that can execute a set of heterogenous video applications on a multi-core processor system in a controlled way to optimize efficiency and/or quality. In many applications and cases, cost constraints are imposed on the system, so that computing resources are inherently limited. If the amount of applications and their load is too high for the system, there is no other way then to control the allocation of resources to individual processing tasks. The optimization of quality for a set of parallel video tasks is carried out by a QoS unit. For a good QoS control, performance analysis and prediction is indispensable. If the application tasks are well modeled and their required performance is known in the form of computation, memory and bandwidth budgets, the platform can be efficiently filled with a suitable set of applications. One trend makes the concept of performance analysis and QoS more complicated: video applications tend to be less streaming oriented and increasingly make use of analysis of the data and specific features contained within the data for further processing steps. This trend not only occurs at the application level, but also in general video processing, i.e. to extract features about the content of the video so that processing becomes aware of the type of data that is being processed. In addition, the nature of analysis applications is more dynamic in its behavior with respect to both computing and memory usage, when compared with streaming and regular video processing.

Summarizing the problem statement is that given a multi-core system, how should a dynamic video application consisting of multiple tasks be efficiently mapped on the system? Furthermore, if the application desires a sudden change in functionality or an increase in complexity, how do we reconfigure the mapping of the video application into another efficient solution? Finally, can we consider general quality attributes as image quality, throughput or latency in this optimization and what techniques can be used to consider this optimization?

1.3 Case study applied in this paper

Our case is based on the medical imaging and analysis, in particular on advanced enhancement processing with strict latency constraints. Region-based image analysis contains advanced and conditional processing. These processing tasks occur also in intelligent surveillance systems and many other video content analysis applications such as face recognition, etc.

With minimal invasive interventions, cardiologists diagnose and treat coronary artery diseases using a catheter inserted into the groin and threaded through the arterial vessel tree to reach the heart [22]. Radiologists diagnose and treat vascular stenosis, thromboses and aneurysms by inserting catheters in the veins [20].

Coronary angioplasty is a catheter-based procedure performed by an interventional cardiologist to open up a blocked coronary artery and restore blood flow to the heart muscle [27]. The analysis and motion compensation techniques can improve the visualization and measurement of objects of interest (such as stents) in real-time X-ray interventional imaging, thereby making it easier to realize optimum and complete stent placement. We focus on a medical-imaging application to detect and enhance moving features, combined with a second pipeline of tasks for increasing the image quality during a live interventional angioplasty procedure [3], (Fig. 1a).

The case study involving image analysis and motion-compensation tasks is used to detect and enhance objects of interest (Fig. 1b). Although a first branch contains stream-oriented tasks having a static nature in terms of computations and memory, a second branch contains data-dependent dynamic processing tasks, involving image analysis, which has a more dynamic nature than streaming video. The dynamics come from properties like the variable size of the analysis region, the feature detection and the dynamic abortion of non-appropriate tasks if the detection or registration result is insufficient.

1.4 Paper overview

The paper is organized as follows. In Sect. 2, we propose a performance-analysis technique leading to a prediction model for computing tasks. In Sect. 9, the memory and bandwidth usage is analyzed and modeled. The combination of computation and memory bandwidth modeling allows us to analyze the mapping of tasks at a higher level and involve estimation of the complexity. In Sect. 4, a hierarchical QoS framework is introduced, addressing individual task quality optimization and overall performance simultaneously. The section concludes with reconfigurable application flow graphs such that an optimal combination of quality and application functionality is obtained. Section 5 presents conclusions.

2 Performance analysis for computations

2.1 Preliminaries and related work

The above-described application and the dynamic behavior of video analysis tasks in computing lead to more complicated situations for the QoS control unit. For this purpose, we have developed a technique to design computational models for dynamic applications, estimating the computational complexity.

Before presenting an approach, we first classify different performance techniques to come to our proposed modeling technique. This modeling involves several important aspects. First, the signal processing is described by Synchronous Data Flow (SDF) graphs [19]. On these graphs, the processing is executed in the form of jobs and tasks which can have a processing or memory-oriented nature. The complexity of the task is captured by the Worst-Case Execution Time (WCET). If several tasks are executed on the same processor, they can be queued and the involved delay and order can be analyzed with queueing theory. The execution of signal processing tasks can be deterministic or non-deterministic from nature. As the WCET is a worst-case model, it would lead to unused load of resources with data-dependent tasks. Therefore, we also consider statistical techniques for the modeling of the execution. Finally, simulations contribute provide insight about the execution and the model accuracy during the execution. Let us now discuss this in more detail.

- *Analytical methods based on queueing networks*. The execution of a platform is described by servers and jobs. Jobs are first inserted into queues and waiting until the server can handle their requests. A job is characterized with an arrival rate, a queue by an average number of jobs in the queue, and a server with the mean service time. The platform can then be described and analyzed as an M/M/1 queueing problem [18]. Because this modeling applies to general purpose computing of tasks, modeling stream-oriented processing of chains of functions with a high accuracy is difficult.

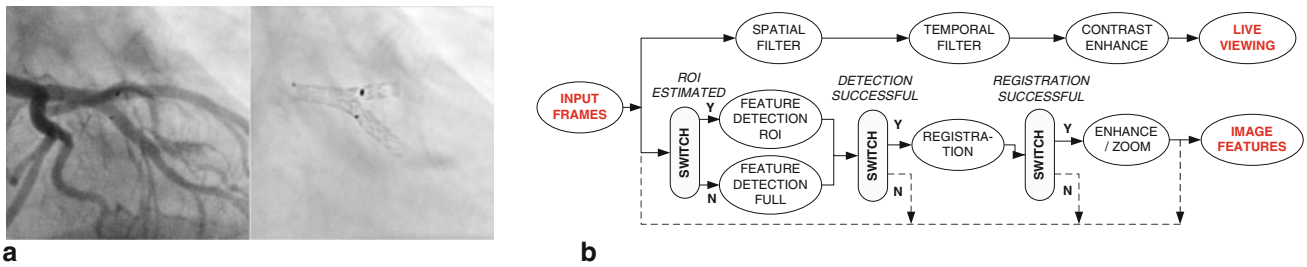


Fig. 1 **a** Coronary angiogram of a stented bifurcation and the view of the enhanced stent. **b** Flow graph of an interventional X-ray application to detect and enhance moving features, combined with several image quality functions

- *Statistical techniques*. These techniques are data-driven approaches based on the input data characteristics. Popular techniques are linear models [16], stochastic models [38], or Markov-chain analysis [8, 40]. The advantage of such models are the abstraction from irrelevant issues at different stages of the design, and very fast estimation of the execution behavior. This type of modeling is particularly attractive for processing with data dependencies or dynamics within the processing tasks itself, leading to considerable variations in execution time and behavior. Statistical techniques are then used to describe these variations in a model.
- *Simulations*. One of the mostly employed techniques, which is based on the construction of a simulation model that is typically executed on a host computer system [12]. It is used when the analytical methods do not allow the use of previous methods, because of the complexity of the exploration space. The results are highly dependent on a proper selection of the input data. Prior work by Pastrnak and de With [28] modeled the execution of multimedia streaming tasks in linear timing equations, where possible data dependencies are handled by including important coding parameters into the model.

When considering the above classification, the problem of performance analysis for dynamic video processing applications is only partially suited for queueing theory approaches. It will give a model that is not accurate enough for the behavior prediction of dynamic tasks, when the objects are relatively small, leading to unpredictable statistics. For dynamic processing, we have to adopt another method, as the above approach will fail in case of dependencies and dynamic behavior in processing. Statistical techniques describing the stochastic nature of the execution seems more elegant, because the data dependencies need not to be known in detail to construct a feasible model. Simulation and modeling play an important role for model construction and tuning to sufficient accuracy. In our case, we can distinguish tasks that are fully independent of the input data and also tasks of which the complexity is highly dependent on the input. Furthermore, tasks exist for which the algorithm switches between scenarios, leading to a highly varying computation behavior. The case that we will present is dynamic (and data dependent) with the following characteristics:

- dynamic decision making in the flow graph is based on outcomes of the analysis tasks,
- the flow graph may switch to a different group of tasks, based on the image content,
- tasks cannot be easily switched off, as this will lead to an unacceptable processing result.

In the next subsection, we develop a concept for performance analysis and prediction of resource usage, which addresses each of the previous aspects individually to closely follow the statistical variations in computing requirements.

2.2 Performance analysis and prediction employing statistical techniques

Approach. Prior to developing the actual model, we first introduce a design flow for the resource-usage estimation process (Fig. 2). The input from the application is in the form of an executable specification. The specification distinguishes individual processing jobs and each job is divided into tasks. Note that for the mapping of a set of tasks onto a multi-core platform, the assignment of tasks to a distinct processor core requires advanced mechanisms (such as floating tasks in the form of actors [28], and a runtime framework such as described in [35], which will not be addressed here. The hardware architecture is represented by accurate simulation models, e.g. an instruction-set simulator for a processing cores. At the timing analysis stage, each job is characterized by application-specific performance constraints on the throughput of the job. The complete application activates a sequence of jobs. Each job can use multiple processing cores in parallel. We model a job as an iterative "for" loop, taking J iterations to produce J data tokens of a video data block. The body of the "for"

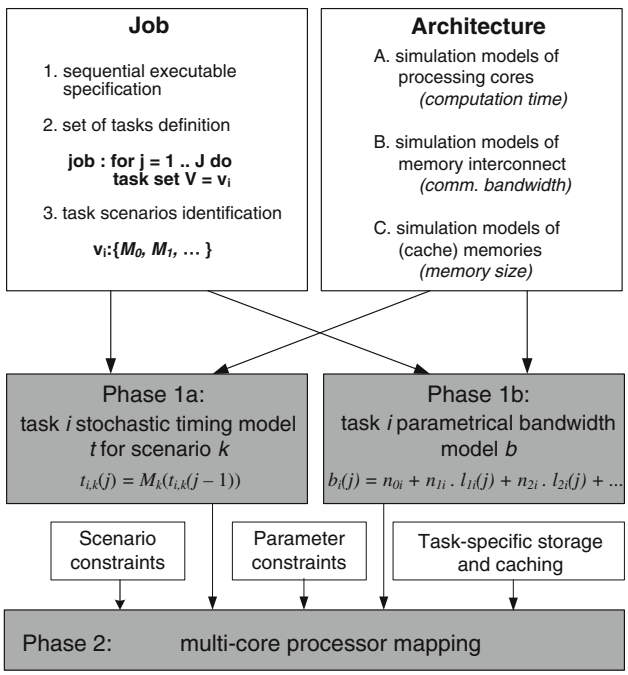


Fig. 2 Design flow for the resource-usage estimation process (Phase 1a) and memory and bandwidth analysis (Phase 1b)

loop in Fig. 2 is described by tasks and the dependencies between them.

We denote the set of tasks as $V = \{v_i\}$. An actor can be described as a function in a programming language like C/C++, representing an atomic unit of computation that can be assigned to a processing core. We assume that the firing delay of a task v_i can be expressed as a linear timing function t_i. As motivated earlier, we propose to model the computation resources for task i and scenario k in job j giving computation time $t_{i,k}(j)$, specified by $t_{i,k}(j) = M_{i,k}(t_{i,k}(j-1))$, with M_k, the stochastic model for scenario k, depending on the timing results from the previous job iteration. Note that both the values of the individual timing functions and the task scenarios may change in each iteration of the "for" loop.

Let us now continue to develop the statistical model for the computing behavior of the jobs and tasks as defined above. Dynamic behavior generally points to the direction of creating a state-space diagram to capture al the dynamic transitions in the processing. A deterministic state-space covering all the dynamics with sufficient accuracy will have two problems. First, the state space will grow exponentially when adding more algorithm characteristics to the model, which can cause memory or computation problems.

Second, another problem is to obtain statistical significant estimates for the transitions between states. The state space becomes large with an increasing number of states, and the number of samples for each state may become very small, even if we use very long training data sets. Therefore, we have decided that the modeling should be based on stochastic properties and it should be able to handle abrupt changes in behavior and statistics.

To illustrate our solution direction, we have measured the computation behavior of a representative task in the dynamic image analysis part of the case study. The result of this dynamic behavior is shown in Fig. 3. The complete case involves several of such dynamic tasks and various branching situations. An alternative view on the modeling of the system behavior is to consider the timing statistics of the video frames and classifying the statistics in categories. The figure suggests to consider trend-based changes independently from the local variations in computing requirements. Let us investigate short-term and structural fluctuations in processing time on the platform.

Short-term fluctuations can be caused by cache misses or the overhead imposed by task switching and control. Structural or long-term fluctuations are caused by the dependency of the processing time of the tasks on the video content itself over a longer time period. This motivates the splitting of the computational statistics in categories and employing different models for those categories.

- *Short-term data correlations*. We have investigated literature on video traffic modeling [13]. An approach resulting from the literature is the creation of a Markov chain, since the estimation of the model parameters is straightforward and various analysis techniques are available. A first-order Markov chain is by definition memoryless, where in the model it is implicitly assumed that the processing times of successive frames are independent. However, Markov chain prediction falls short if processing times between video frames are correlated over a longer time period. Next, we will describe the modeling of long-term structural data dependencies.
- *Long-term data correlations*. We consider the prediction model to consist of long-term low-frequency fluctuations, around which short-term high-frequency fluctuations can take place. Discrimination between the two frequency parts can be made by various types of filters, such as Finite Impulse Response (FIR) or Infinite Impulse Response (IIR) filters. In Fig. 3, this is illustrated when applying a Low-Pass Filter (LPF) and a High-Pass Filter (HPF) to a given input signal.

Let us now further analyze the meaning of Fig. 3 and concentrate on the short-term statistics (Fig. 3b). Given the separation of correlation behavior, the short-term fluctuations are modeled with Markov chains. We have validated the applicability of the Markov chain modeling by

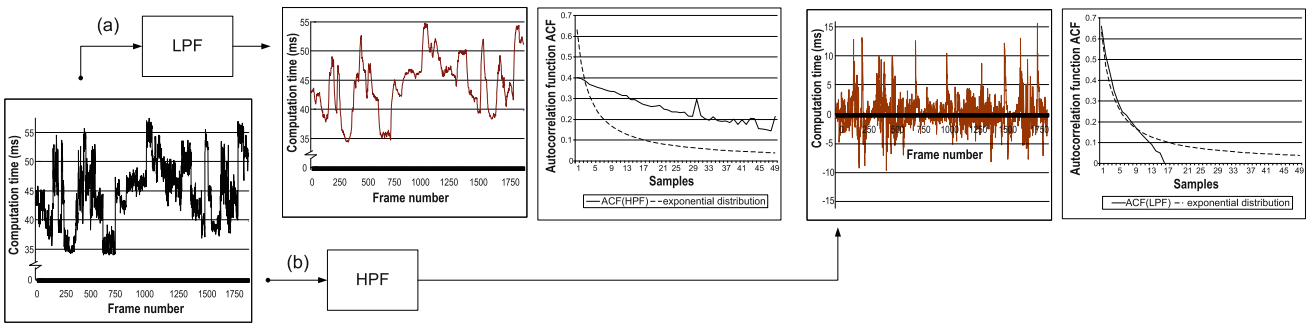


Fig. 3 Decoupling **a** the long-term from **b** the short-term statistics with Low-Pass (LPF) and High-Pass Filters (HPF)

analyzing the autocorrelation function ACF. As the function has an exponential decay, Markov chain analysis is applicable.

The state-space description of an application can be generated by analyzing the computation time C over a long time period. The number of states evolves from analyzing the computation behavior function and quantizing this function into categories becoming Markov states, based on the number of samples in that category. More specifically, the number of states M is $C_{\max}/\sigma_C$, where $C_{\max}$ denotes the largest measured value and σ_C the standard deviation. We have experimentally evolved to a model with approximately $2M$ states to obtain sufficient accuracy. The quantization intervals are adaptively chosen such that each interval contains on the average the same amount of samples. The entries of the transition probability matrix P_{ij} are now estimated by

$$P_{ij} \simeq n_{ij} \Big/ \left(\sum_{k=1}^{M} n_{ik} \right) \tag{1}$$

where n_{ij} denotes the number of transitions from interval i to interval j. The short-term behavior is now emulated by allowing a random variable to be executed according to the specified Markov source, based on the actually measured value of the computation complexity within the preceding interval.

Besides the modeling of short-term behavior, long-term behavior prediction is based on the statistical filter theory, as mentioned earlier. We have adopted a moving average filter for this purpose, specifically the Exponentially Weighted Moving Average (EWMA) filter. The starting state is the preceding interval where the computation complexity parameter is used as an input to the filter. The output of the filter then produces the prediction for the upcoming interval. As this Infinite Impulse Response (IIR) filter weights recent inputs more heavily than long-term previous ones, it adapts more quickly to the input signal compared with FIR filters. The EWMA filter is defined by

$$y(t_k) = (1 - \alpha) y(t_{k-1}) + \alpha x(t_k). \tag{2}$$

Because we have now covered both short-term and long-term statistics, the computation time for the current video frame can be predicted using a combination of both approaches as suggested by Fig. 3. More details will be provided after the presentation of the case study that was used to validate the above techniques.

Case description. The presented application for motion-compensated feature enhancement consists of several steps, as shown in Fig. 4. As mentioned earlier, the application contains several dynamic tasks, based on analysis, in combination with conditional switch statements. The presented flow graph is based on a cascading of four stages which are individually described in [9, 10, 11, 36]. After stent placement, the marker candidates are detected in the image using an automatic marker extraction algorithm. Ridge detection (RDG) and filtering is applied to the input images such that all structures except marker candidates are removed.

Figure 3 (left) shows the computation–time statistics for the ridge detection (RDG FULL) task, as a representative example of the more variable nature of the computing statistics. Subsequently, marker extraction (MKX EXT) selects punctual dark zones contrasting with a brighter background, as candidate markers. Based on a priori known distances between the balloon markers, couples selection (CPLS SELECT) selects the best marker couple from the set of candidate couples. The guide wire is detected by a ridge filter in guide-wire extraction (GW EXT). If the markers of a possible couple are situated on a track corresponding to a ridge joining them (the guide wire), then an indication occurs that the results obtained by automatic marker extraction are found stable. Subsequently, temporal registration (REG) to align respective markers in selected image frames is based on a motion criterion, where a temporal difference is performed between two succeeding images of the sequence. A region of interest (ROI) is estimated in the original image (ROI EST), where the

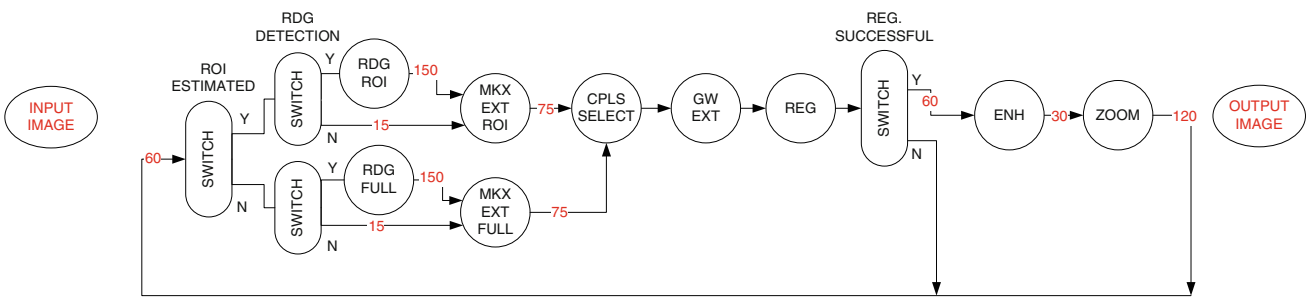


Fig. 4 Detailed flow graph of the interventional X-ray application to detect and enhance moving features [the required inter-task memory bandwidth is shown on the *arrows* between tasks (MB/s)]

markers have previously been detected. Enhancement (ENH) of the stent is performed by temporal integration of the registered image frames according to the markers. The output is presented by zooming (ZOOM) in the ROI containing the stent.

Case analysis. The MKX EXT, REG, ROI estimation, ENH and ZOOM functions are independent of the video content or size of the images. The prediction can be defined with a constant value. There are four data-dependent switch statements in the flow graph. The current state is based on the information from previously processed video frames and can be described with a state table. Each switch can signal tasks to process for example only on a region of interest of the video frame or even skip processing. The RDG, CPLS SELECT and GW EXT functions have a resource usage that is highly dependent on and correlated with the video content. Summarizing, the described application is dynamic in three major aspects: (1) at the start, an ROI of variable data-dependent size is chosen for further analysis, and (2) at every stage, a switch function selects a specific flow graph, depending on the previous stage(s). Moreover, (3) some of the internal flow graphs require intrinsically a variable processing.

Model results. The computation behavior is modeled with the presented approach using long-term and short-term statistics and their corresponding modeling techniques. Supplementary to these techniques, we have found that the computation behavior is dependent on a linear function representing the size of the analyzed part in the image (the ROI size). An additional element is the handling of the switch statements in the flow graph. To come to a realistic modeling result, the state result of the possible switch in the algorithm has been used as pre-knowledge for selecting the correct scenario.

Based on the computation of the autocorrelation function, we have concluded that ridge detection, (RDG), couples selection (CPLS SEL) and guide-wire extraction (GW EXT) tasks can all three be modeled with Markov chains. Data-dependent switch statements in the flow graph can cause the total processing time to change rather abruptly. For example, the first switch in the flow graph can select the RDG task to operate only on an ROI instead of the full video frame. Other switch statements trigger or cancel tasks to be executed. The switches are controlled with information extracted from the previously processed video frames, and stored in a state table. At the start of processing each new video frame, the state is extracted in advance. By exploiting this information prior to the actual processing of the task graph, the prediction model is made adaptive to dynamic changes in the data flow. This part corresponds to the scenario-based switching in [33].

Processing time statistics for different ROI sizes show that the RDG task has a linear dependency on the size of the ROI. To analyze load fluctuations, caused by dependencies on the video content itself, we have subtracted a linear growth function from the obtained statistics. This function is specified by

$$y_{t_k} = 0.067 \times \text{ROISize} + 20.6. \quad (3)$$

For the remaining data-dependent fluctuations after subtraction, we have analyzed the autocorrelation function. As the function has an exponential decay, it can again be described with a Markov chain. Because the fluctuations are in the same order as the high-frequency behavior described earlier, we have included these statistics to the Markov state-generation process, to generate a single Markov chain for the ridge-detection (RDG) task.

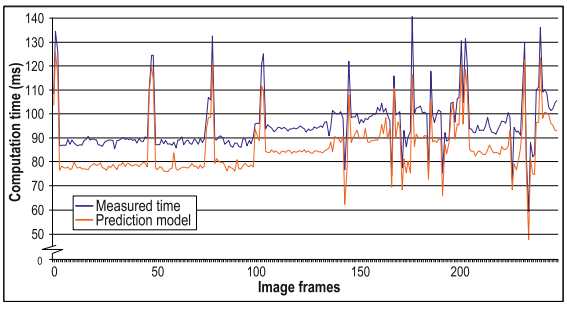
The prediction results for all tasks in the flow graph of Fig. 4, are shown in Fig. 5c. Computation time statistics are obtained by profiling the executed application on a general-purpose multi-core platform [34]. For training the stochastic models, we used a data set of 37 video sequences of in total 1,921 video frames. In the training set, different scenarios exist to which the algorithm tasks are adaptive. In Fig. 5a, the Markov transition matrix is shown for the

	s0	s1	s2	s3	s4	s5	s6	s7	s8	s9
s0	0.51	0.19	0.16	0.03	0.03	0.02	0.01	0.01	0.01	0.01
s1	0.21	0.21	0.17	0.15	0.07	0.08	0.02	0.06	0.00	0.03
s2	0.04	0.08	0.33	0.13	0.16	0.10	0.04	0.06	0.02	0.04
s3	0.04	0.05	0.19	0.16	0.20	0.14	0.06	0.08	0.04	0.04
s4	0.03	0.04	0.16	0.15	0.23	0.12	0.09	0.12	0.03	0.02
s5	0.03	0.05	0.11	0.15	0.18	0.18	0.15	0.09	0.05	0.03
s6	0.02	0.02	0.09	0.08	0.14	0.20	0.14	0.19	0.07	0.04
s7	0.02	0.04	0.08	0.07	0.10	0.09	0.16	0.25	0.12	0.07
s8	0.03	0.01	0.13	0.03	0.09	0.08	0.14	0.25	0.07	0.17
s9	0.01	0.01	0.03	0.01	0.03	0.03	0.06	0.10	0.12	0.60

a

Task	Prediction Model (ms)
RDG FULL	<Eq. 1> + Markov RDG
RDG ROI	<Eq. 3> + Markov RDG
MKX EXT	2.5
CPLS SEL	<Eq. 1> + Markov CPLS
REG	2
ROI EST	1
GW EXT	<Eq. 1> + Markov GW
ENH	24
ZOOM	12.5

b



c

Fig. 5 **a** Markov transition matrix for RDG task and **b** summary of the prediction models and **c** visual representation of the obtained timing model and real execution for 10 test sequences of the full flow graph of Fig. 4 (the prediction is plotted 10 ms lower for improved visibility)

ridge-detection task. Similar Markov transition matrices are generated for the couples selection and guide-wire extraction tasks. A summary of the prediction models can be found in Fig. 5b. To validate the prediction mode, we have applied a set of 10 test sequences. The results show an average prediction accuracy of 97% with sporadic excursions of the prediction error up to 20–30%.

Discussion of the results. It seems interesting to compare the dynamics of the case study to similar published applications, although the numbers are sometimes not available or not suited for a fair comparison. Generic examples can be found in the domain of computer vision algorithms [39]. Within medical imaging, registration and object recognition tasks with similar dynamics are studied for quite some time [23] although these algorithmic tasks are still not readily available in applications with real-time constraints. It is expected that with the continuous increase in computational power, future applications will arise in the real-time video domain [15]. In other applications, for example within surveillance, object-tracking applications having similar dynamics and can be found in [25].

In the next section, we continue our modeling with performance analysis and prediction for communication resources and memory usage.

3 Performance analysis for memory communication bandwidth

3.1 Introduction

Besides the modeling of the computation time, as presented in the previous section, the allocation of memory communication resources is at least as important and motivated as follows.

Multi-core systems are characterized by a set of processing cores on a chip centered around a communication infrastructure, which is connected to a large off-chip memory. For such architectures, it is evident that off-chip memory bandwidth is a critical part for system performance and cost. Chip connections and power consumption for drivers hamper the use of large width buses, so that memory bandwidth cannot grow with the same speed as computation. Communication bandwidth is, therefore, a scarce resource in the system, and efficient usage of the available bandwidth is of equal importance as the use of computation resources.

For many applications in video processing, there is a direct relation between memory usage, memory bandwidth and the algorithm parameters [17]. For example, in the case of MPEG-2 coding, the required amount of memory needed in the search area for motion-vector search depends linearly on the maximum allowed velocity of motion of moving objects. Furthermore, the memory usage for VLC coefficient scanning grows linearly with the amount of coefficients.

Memory requirements for stream-based functions depend on the filter processing aperture. Similarly, for content-analysis applications, the region-of-interest size around objects in feature analysis has a direct relation to the amount of memory and communication required. Summarizing, an estimation model for communication bandwidth will depend linearly on key parameters of each task and the amount of required input data. This reasoning was validated earlier by [28] for a case with object-based MPEG-4 coding. We adopt the approach to use linear models for communication and memory usage and refined it for our case study on medical image analysis. The result is that the design flow for computations from Sect. 2 is reused except for some memory-specific modifications.

Probably, memory-bandwidth analysis has an even more practical impact than computations, because in a multi-core system, bandwidth is becoming increasingly the most scarce resource parameter in the system.

3.2 Memory-bandwidth analysis

Approach. We deploy a cooperative user-level task scheduling framework from literature [35] on top of the operating system. The framework creates only one thread per processor core and one task queue per last level cache memory. Applications are specified as connected (streaming) tasks of a directed dataflow graph, where the input and output buffers of tasks are kept locally in the cache memory of the processor. When an input image is received, the input relations of the first task are satisfied and the task is send to one of the task queues of the scheduling framework. Cooperative task scheduling is preferred when throughput is more important than fast response times.

At compile time, splitting of tasks into subtasks is organized in such a way that all processor cores are active and the load on the system is balanced. Tasks cannot move between processor cores as we did not implement task stealing or dynamic scheduling concepts. Tasks that are scheduled to a task queue for a particular processor core can not move to another task queue. In this way, task data blocks remain local in one of the two L2 caches, which avoid cache misses due to tasks that are moving from one core to the other. The interested reader is referred to [1, 14] for more details.

We denote the set of tasks as $V = \{v_i\}$. The timing properties of the individual tasks need to be completed with the bandwidth requirements between tasks $b_{i,k}(j)$, to provide a complete analysis on a multi-core platform [32]. As

motivated earlier, we propose to model the communication resources with linear parametrical equations, for task i in job j for scenario k giving bandwidth $b_{i,k}(j)$, specified by:

$$b_{i,k}(j) = n_{0,i} + n_{1,i}l_{1,i}(j) + n_{2,i}l_{2,i}(j) + \cdots. \quad (4)$$

The variable n_i stands for the weighting coefficients of the term contributing to the communication, and the variable l_i denotes a specific input-data parameter, such as the size of the ROI in the image. This specification completes the analysis model. The modeling of bandwidth is defined as the total amount of input and output memory traffic for all tasks between the external memory and the last level cache memory near the processor cores. As some of the tasks require more memory intrinsically than available in the cache, additional memory traffic is generated due to cache misses. This aspect is modeled as a separate factor, and then added to the input/output memory bandwidth.

Case analysis. When analyzing the case study used throughout this paper, the required amount of memory for each task can be derived by extracting the input/output requirements and intermediate storage requirements from a reference software implementation. In Table 1, the results are shown for the RDG, MKX, ENH and ZOOM task. For concentrating on significant bandwidth requirements, only operations on arrays are taken into consideration. The tasks that operate on a subset or feature data are negligible in terms of memory consumption. Note that some of the functions have different memory requirements, depending on the state of a switch in the flow graph. For example, if the RDG task is switched off, the succeeding MKX function has a much smaller input buffer requirement. The amount of communication bandwidth required for correct operation of the image analysis application is derived by a mapping of the memory requirements onto the platform architecture.

To calculate the amount of memory communication bandwidth of the application under study, there are a number of constraints that influence the actually measured bandwidth load on the platform architecture.

Hardware platform. The choice for a particular platform sets an upper limit on the available resources. For the experiments, we use a general-purpose multi-core platform containing of two quad-core (Xeon) processors. In Fig. 6, the system is shown. In total, the system consists of eight processors of 2.33 GCycles/s, eight level-1 caches of 32 kB and four level-2 caches of 4 MB. The system is equipped with 4 GB of external memory. For more details about the instantiated architecture, we refer to [34].

Application scenarios. The memory-bandwidth requirements vary continuously by switching between different

Table 1 Memory requirements for each task of Fig. 4

Task	RDG select	Input (kB)	Intermediate (kB)	Output (kB)
RDG FULL		2,048	7,168	5,120
RDG ROI		2,048	5,120	5,120
MKX FULL	–	512	512	2,560
MKX ROI	–	512	512	2,560
MKX FULL	X	4,608	512	2,560
MKX ROI	X	4,608	512	2,560
ENH		2,048	8,192	1,024
ZOOM		1,024	4,096	4,096

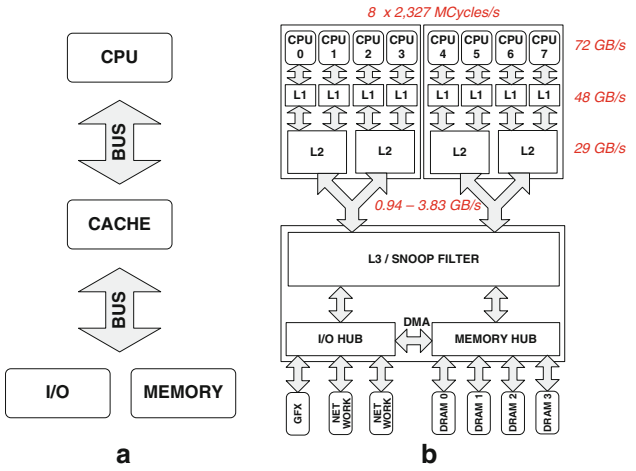


Fig. 6 **a** Generic architecture model, **b** instantiated architecture with parameters

groups of processing tasks (scenarios). For the worst-case scenario in terms of bandwidth requirements, the tasks operate on a full-frame granularity, the ridge-detection task is activated and the registration phase completes successfully. On the opposite, for the best-case scenario in terms of bandwidth requirements, the tasks operate on a small ROI granularity, which will save a significant amount of required bandwidth. Furthermore, if the ridge-detection task is not required and the registration is not successful, the enhancement and zoom tasks will be skipped, which will save another significant amount of required bandwidth. Note that in this "best-case" scenario in terms of bandwidth, the algorithm will not output a satisfying result. In total, there are eight different scenarios possible, given the three switch statements in the flow graph.

Intra-task memory. If a task internally requires more memory that can be stored locally in the cache memory of the processor, additional memory bandwidth will be

initiated to swap data in and out the local cache memory. For the application under study, the RDG, ENH and ZOOM tasks have an intra-task memory requirement that is higher than the level-2 cache capacity of the processor. The amount of bandwidth of each task can be modeled by analysis of the access pattern of the internal buffers. As the above tasks operate on a whole image granularity, scanning of the pixels linearly in the (x, y) direction will enable that all data items will be accessed.

Inter-task memory. The partitioning of the application on the platform has a direct relationship with the required amount of memory bandwidth between tasks. In Fig. 4, the required inter-task bandwidth is shown on the arrows between tasks (1,024 × 1,024 pixels, 30 Hz, 2 Bytes/pixel).

Results. Memory communication bandwidth is estimated for each task by analysis of the cache-memory usage for the set of test sequences. For each task, we start with the analysis of the input and output memory requirements. This sets a lower bound of the required memory communication bandwidth between the external memory and the local cache near the processor cores. As an example, for the RDG task, at the input, 2,048 kB of memory is required, where at the output, 5,120 kB is required. The required memory bandwidth, counting only input and output buffer requirement is, therefore, already more than 100 MB/s, when processing images at 15 Hz. In addition, some of the tasks internally require more memory that can be stored locally in the cache memory of the processor. This additional bandwidth that needs to be initiated to swap data in and out the cache memory is modeled separately. The RDG task consists of 3 subtasks, where in total 13 intermediate storage buffers are required of in total 5,120 kB. As the size of the local cache memory is 4,096 kB, shared between two processor cores, the required memory usage will overload the cache memory, which obviously results in additional memory communication bandwidth to and from external memory.

We model the cache-memory usage and corresponding load/store behavior with space–time buffer occupation models. As a representative example, we describe the modeling for the RDG task in more detail. The load/store pattern between external memory and cache memory is shown in Fig. 7. Although the access pattern of the storage buffers is linear, some of the internal buffers are required more than once, which requires some additional bookkeeping to keep track of the amount of bandwidth that will be generated due to the limited cache capacity. The memory bandwidth between external memory and the local cache of the processor is estimated by the amount and size of the temporary buffers that are loaded, flushed and

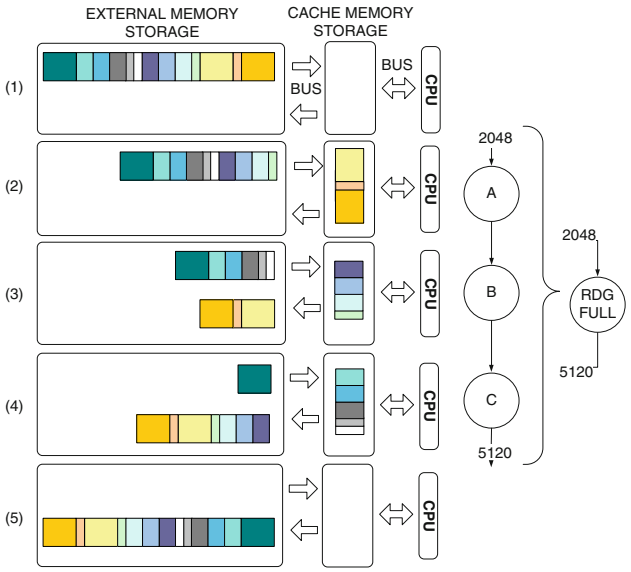


Fig. 7 Required intra-task bandwidth for RDG FULL task due to the limited cache-memory storage

reloaded from external memory, at the start of processing each subtask (stages 2–4 in Fig. 7).

Summarizing, the complete model for the memory bandwidth usage is created by adding the bandwidth requirements of intermediate storage to the input/output bandwidth requirements, as explained in the beginning of this paragraph. We have compared the model output values with the actually used bandwidth for execution of the image analysis application. At a scenario level, the memory resource usage is more or less constant. The size of the ROI only slightly impacts the memory usage, therefore, the differences between a large ROI and a small ROI are negligible in terms of bandwidth.

For the test sequences, an average accuracy between the analysis and measured memory bandwidth usage of 90% is obtained. A limitation of the preceding experiment is that the model is relatively static and does not address the use of bandwidth fluctuating functions when not captured by a few scenarios. For such a case, we propose to explore the techniques which have been employed for computation analysis and we would model statistical analysis of the memory bandwidth. This solution is not further elaborated in this paper as the processing time of an image is by far dominated by the computational complexity of tasks, but the authors expect that this approach would be equally applicable for modeling memory-bandwidth usage.

In the next section, we introduce a layered QoS architecture, employing the prediction models from the last two sections for the predicable mapping and execution of multiple dynamic video processing tasks on a multi-core processor architecture.

4 Runtime QoS and task-level scalability

4.1 Introduction

For applications sharing limited platform resources, there will be system constraints on the available computing power, memory and bandwidth. If the computational load becomes too high for the platform, we intend to reduce the effort involved for particular tasks without the complete abortion of the job execution. To do this in a controlled way, we need two elements. First, the applications should have scalable properties in terms of performance and computational effort.

Second, the platform should be able to control and monitor a number of parallel applications and their resource usage. Unfortunately, this overall control task cannot be carried out by a conventional operating system, because it lacks the knowledge about the desired overall system usage and it has no understanding about the impact of quality settings for the individual applications and the overall quality of the complete system. The conclusion of this discussion is to implement a special QoS system that is capable of handling the control aspects dealing with multiple applications.

QoS control has been subject of research for a number of years, for e.g. MPEG-4 3D graphics, wavelet coding, picture in picture video, etc., see [4, 7]. Thus far, this work has focused on scaling the application and then elegantly distributing the computing tasks on a single processor system. Therefore, we concentrate on the part that is usually missing in this type of research. In the case study used throughout this paper, we discuss multiple dynamic tasks running in parallel on a multi-core processor platform. This covers the cases of executing a single advanced application or a set of applications in parallel. This problem statement is a new element and distinguishes itself from previous work on QoS management. Therefore, we focus particularly on the video application part of the overall resource management.

To allow both single and multiple video applications in parallel, which should be controlled with the same QoS concept, we adopt the *hierarchical* QoS architecture from [30]. In this architecture, a Local QoS (LQoS) controls an individual application while a Global QoS (GQoS) controls the complete set of active applications and optimizes the overall system behavior. In the sequel, the term "quality" in the discussion on system performance control should be interpreted broadly. Hence, not only image quality, but also other performance critical parameters, such as the mentioned latency and throughput can be involved.

The next subsection provides more details on the hierarchical concept together with a model for runtime resource management. Afterwards, several options for task level scalability are presented including QoS reconfiguration experiments and results.

4.2 Hierarchical QoS architecture

The hierarchical QoS system addresses both the optimization of resource allocation for individual applications and for a complete set of applications. We will not discuss QoS service algorithms in high detail. Instead, we assume that an algorithm from literature can be adopted, given the broad availability of proposals on this topic [2]. The architecture has a hierarchical, layered structure. It consists of two communicating managers, instead of the conventional single resource manager. The layered approach separates the system control optimizing overall quality and behavior from the responsibilities of individual application QoS units. The responsibilities of the two QoS managers are as follows.

- *GQoS manager*. It controls the total system performance involving all applications running in parallel. This manager optimizes the user benefits, such as the available parallelism and priorities of applications, instead of a single video application.
- *LQoS manager*. It controls an individual application within the assigned resources, which were provided by the GQoS manager. This manager optimizes the individual application quality for the agreed amount of resources.

An overview of the system architecture and QoS control is shown in Fig. 8a and b, respectively. Each application is divided into jobs and the platform supports the execution of each job. Each individual application is controlled by a LQoS unit, which negotiates about the assigned resources with the GQoS control. After negotiation, the LQoS unit has an exact specification of the amount of resources that are assigned to the application. The GQoS unit depends on the resource manager for the actual resource assignment. The resource manager provides the real allocation of physical resources in conjunction with an operating system. The operating system is used for allocating tasks to threads, where each thread is fixed to a processor core. The control task then manages which tasks are running at which quality setting. This is a new aspect that is added on top of a conventional operating system.

The control of an application execution is conceptually visualized in Fig. 8b. The execution starts with the activation of the LQoS unit. The LQoS unit activates an internal resource estimator, which calculates the resource usage requirements at all quality levels of the required jobs. In Step 2, the query is send to the GQoS unit, which decides on the highest quality level that can be guaranteed

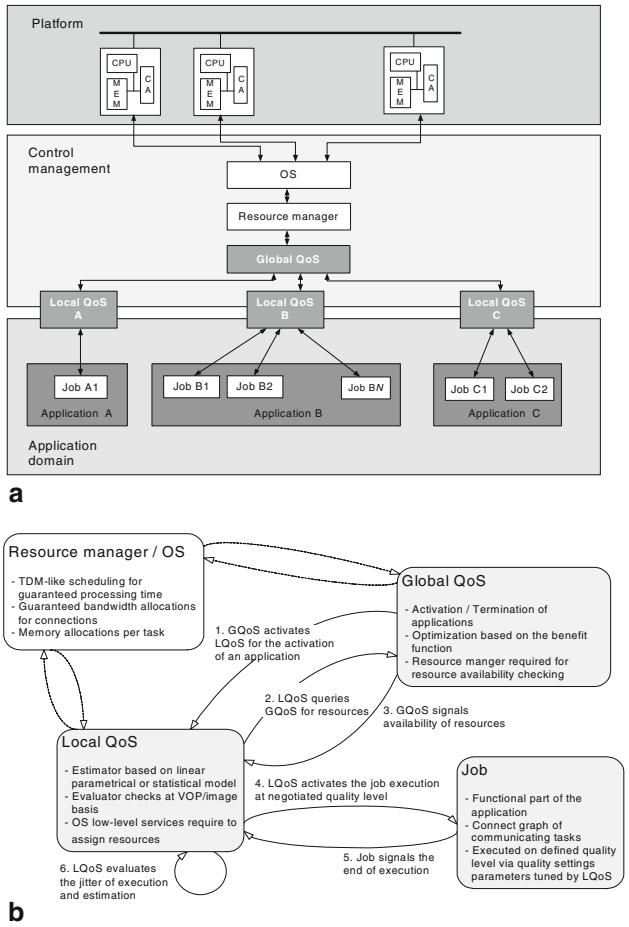


Fig. 8 **a** Layered view of the system architecture and **b** execution of a job in the system with the layered QoS control

and allocated for the application. In Step 3, the response is communicated back to the LQoS unit. In the case that resources can be assigned, the LQoS unit allocates individual tasks of the job at the chosen quality and invokes the job execution in Step 4. In Step 5, the job signals the end of the processing period. In the medical imaging case study, this means the completion of the enhancement and zooming task. In Step 6, the LQoS unit evaluates the difference between the estimation and real execution and in the case the difference is above a predetermined threshold, the LQoS adapts the estimation model. If this does not sufficiently lower the difference, a new negotiation on resources has to be performed (which involves restarting the process from Step 2 onwards). The estimation of total resources is based on the number of jobs and their individual estimates on resource usage.

QoS problem definition and applied heuristic. Let us now specify the QoS problem in a more formal way. We describe the resource requirements of the job i at all defined quality settings (vector q_i) per resource J by

$$R_{i,J}(\mathbf{q}_i) = f_J(\mathbf{q}_i). \tag{5}$$

The resource type is $\mathbf{J} \in \{\mathbf{C}, \mathbf{D}, \mathbf{B}\}$, where $\mathbf{C}$ denotes the computation resources per task, $\mathbf{D}$ the data memory per task, $\mathbf{B}$ the required bandwidth. The optimization problem for our GQoS management can be formulated as maximizing the overall quality, specified by

$$\max \sum_{i=0}^{N} \beta_i(\mathbf{q}_i(c)), \tag{6}$$

with chosen quality $\mathbf{q}_i(c)$, subject to

$$\sum_{i=1}^{N} R_{i,J}(\mathbf{q}_i(c)) < \sum_{j=1}^{M} P_J(j), \text{ with } \mathbf{J} \in \{\mathbf{C}, \mathbf{D}, \mathbf{B}\}. \tag{7}$$

In the above equation, N denotes the number of jobs and M indicates the number of processing cores. We define the benefit $\beta_i(\mathbf{q}_i(c))$, as the contribution of job i to the user benefit, at selected quality level c, giving the quality $\mathbf{q}_i(c)$. The solution to the above problem statement relates to the 0-1 Knapsack Problem [21] which is an NP-hard problem and cannot be implemented at runtime. In literature, several approaches can be found for solving multi-dimensional resource allocation problems, based e.g. on Lagrangian relaxation or Pareto algebra [26, 43]. In our case, we use a heuristic to compute a near-optimal solution at runtime.

For simplicity, we assume that the system is in operation and has a set of running jobs. We consider four types of possible situations in the running system: (1) job fires a request to be started, (2) job requires more resources, (3) job is terminated, (4) job releases (a part of) its resources. The second type is a special case of the first, and similarly, the fourth type is a special case of the third. Let us further describe the algorithm of the negotiation process for a request to start a new job. The algorithm is detailed in Fig. 9 and parametric descriptions are found in Table 2.

Algorithm 1 Heuristic QoS optimization

```
1: procedure MappingJob()
2: while not(IsEnoughResources(CandidateJob)) do
3:   JobToDecrease = FindMinBenefit(ActiveJobs + CandidateJob);
4:   if JobToDecrease! = CandidateJob then
5:     SetQualityLevel(JobToDecrease, NewQuality)
6:   else
7:     LowerQualityOfCandidate()
8:     if CandidateQuality < Minimum then
9:       return
10:    end if
11:  end if
12:  if IsEnoughResources(CandidateJob) then
13:    MapJob(CandidateJob)
14:    return
15:  end if
16: end while
17: if not(IsEnoughResources(CandidateJob)) then
18:   ReportInsufficientResources()
19: end if
20: endprocedure
```

Fig. 9 Heuristic QoS optimization algorithm

Table 2 Definition of GQoS functions

Function	Description
IsEnoughResource	The function checks if the system can reserve resources at the required benefit $\beta_i(\mathbf{q}_i(c))$. It returns **true** if the allocation is possible.
FindMinBenefit	The function search for a job that contributes with the lowest benefit β_i to the overall system value, returns the job index with lowest benefit increase.
SetQualityLevel	The function sets the quality level to the job. The function is called only for the jobs having a lower benefit than the actual benefit level and it is assumed that a lower benefit requires less resources.
LowerQualityOfCandidate	The function decreases the benefit level of a candidate job, the lowest level is 0.
MapJob	The function calls the *Resource manager* routines to allocate resources defined by the quality benefit level of a job.
ReportInsuffcientResources	The function reports to the LQoS of a candidate job its inability of job execution.

The algorithm is a simplified version of checking the availability of resources for the chosen quality of a candidate Job. At a job analysis phase, appropriate benefits and quality levels have to be assigned. Consequently, the GQoS has defined a benefit for each job, executing on the platform for every possible quality level of a job. The algorithm starts by checking the ability of adding the candidate job to the list of active jobs. The resource estimator within the LQoS manager of a candidate job calculates the required resources. In the case that there are not enough resources, a search for the minimum quality decrease of active jobs to the overall cost function is performed. For an optimal implementation, the GQoS is storing a sorted list of such quality changes, which limits the searching algorithm for finding a new maximum to linear complexity. The algorithm decreases the quality of the set of jobs by reducing the quality of individual jobs, followed by checking whether the system has sufficient resources for a candidate job. The algorithm ends when the new job has sufficient resources for executing the new set of jobs. If the benefit cost function drops below the level at which the system tried to activate the candidate job, the algorithm also terminates.

Due to an increase in resource usage demands for active jobs of application X, an undesired situation may occur when the GQoS assigns only a very limited amount of resources to jobs of application Y. When the LQoS has no quality level that can be satisfied within the given resource constraint, line 17 of Fig. 9 will report insufficient resources, and the job of application Y cannot be executed. As this behavior is not desired, the designer of the system should take care that there are sufficient platform resources available for applications, or sufficient quality levels of tasks.

In case that the platform releases more free resources by ending some of the jobs or changing jobs requirements depending on the input data or a user interaction, we introduce the following strategy. The algorithm starts with a job that gives the highest benefit increase and checks if it can increase a quality level by using new available resources. For efficiency, the system has information about the minimum resources that can increase at least one quality level of a job. If available resources are below this level, the negotiation algorithm stops (the algorithm stops only when it checked all active jobs).

4.3 QoS control at the application: task-level scalability

Introduction. In this section, we change our attention from the platform extensions to the application domain. Although advanced video and imaging applications are typically adaptive to many cases, they are not intrinsically developed to support QoS control. In the literature discussion below, it becomes clear that it is not easy to design a fully complexity-scalable application. For this reason, we concentrate on QoS control at the task level.

We have considered several options for implementing scalability in applications. The approaches found in literature vary between approaches for quality scalability [37], complexity scalability, [24, 41], or coding scalability [42]. The most attractive approach for solving our problem statement is using *complexity* scalability, but the design of a fully complexity-scalable algorithm is omitted here for reasons as discussed above. Instead, we have defined a new type of task scalability based on enabling, disabling, splitting and merging of tasks that compose a job. This form of scalability is much easier to implement and requires only knowledge of the algorithm at the task level. Based on the importance of the task processing to the overall processing, we distinguish *essential* tasks and *non-essential* tasks. Consequently, essential tasks will be given higher priority to resources than non-essential tasks.

For introducing QoS at the task level, it is important to identify scalability options for each task. If a job contains tasks that may be completely idle, and in consequence, the corresponding communication resources are idle as well, they are denoted by dotted lines and gray circles (filter and feature detection tasks in Fig. 10). In the sequel of this

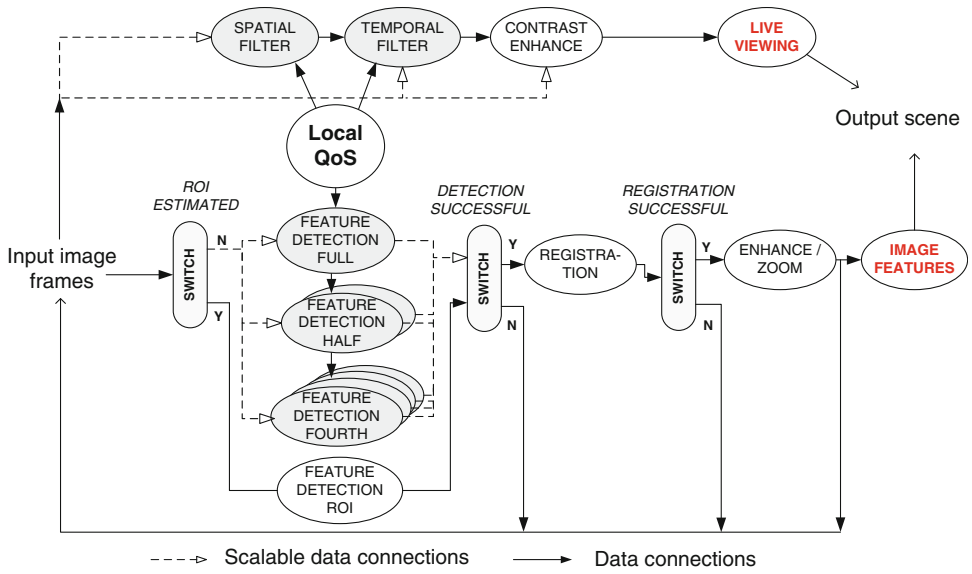


Fig. 10 Flow graph of the medical imaging scene with task-level scalability [task splitting for image analysis (*bottom*) and task skipping for live viewing (*top*)]. The *dotted lines* are optionally enabled or disabled

paper, we report on experimenting with the presented heuristic optimization algorithm to optimize latency and throughput within a professional medical imaging application.

Scalability with task skipping. We have implemented task skipping in the live viewing stream to illustrate task-level scalability for non-essential tasks. The live viewing tasks can be pushed to a lower quality level or even skipped to save (a part of the) assigned resources for other tasks, like the detection of features of interest in image analysis. A second example, which is more relevant for low-cost systems is the possibility to add functionality on a compute platform with a more constrained amount of processing resources (e.g. mobile systems). In more detail for our case study, when image analysis is the main application of interest and not all resources are allocated, the live viewing stream can be enabled as well, with a quality-controlled execution of the image-enhancement tasks. In earlier work [1], we presented a resource-usage model for streaming tasks. This model is useful when the mapping requires multiple processor cores. We have defined the following three quality levels Q of our experimental medical image enhancement application:

– Q1: Basic quality and low resource demand; only contrast enhancement is applied.
– Q2: High quality and resource demand; constrast enhancement and spatial filtering are applied.
– Q3: Highest quality and resource-usage demand; the complete chain is executed.

Scalability with task splitting. As task skipping can sometimes be too coarse in terms of impact on image quality, or the impact on the release or fetching of platform resources, one can follow also a more selective approach, where the task may be split into sub-tasks which can be distributed more easily over multi-core processors. Besides this, it also solves another problem: skipping essential tasks is not acceptable, since it would violate the desired functionality or the quality too much.

With task-splitting scalability for essential tasks, the dynamics in the latency and throughput can be minimized by incidentally reconfiguring the computing tasks in the flow graph, i.e. freeing or consuming some of the resources that were prior budgeted for background tasks. Figure 10 outlines a task-level scalable version of the computation required for the motion-compensated feature enhancement application. Scalability at the task level is achieved by splitting the group of feature detection tasks (RDG and MKX EXT) into sub-tasks, resulting in a lower end-to-end latency. We have defined the following three quality levels Q of our experimental medical image analysis application.

- Q1: Low resource-usage demand; feature detection is executed sequentially.
- Q2: Medium resource-usage demand; feature detection is partitioned into two sub-tasks.
- Q3: High resource-usage demand; feature detection is partitioned into four sub-tasks.

Summarizing the above two concepts for task splitting and skipping, it results in an improved scalability with respect to QoS control and a more fine-grained distribution of tasks over a multi-core processor system.

Experimental results. We illustrate the hierarchical QoS concept in a real-live example. The use case on professional medical imaging is graphically shown in Fig. 11a. The highest priority is assigned to the branch "Image

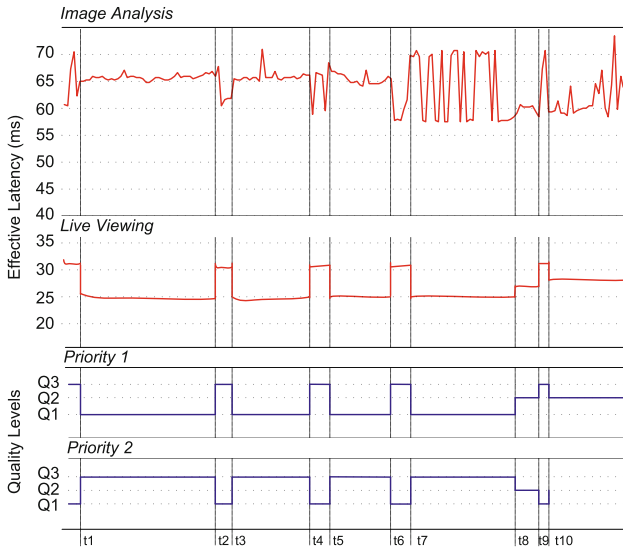


Fig. 11 Results of two concurrently running medical imaging applications with indicated image latencies and quality

Analysis" (bottom branch in Fig. 1). The computational complexity of the analysis tasks depends on the size of the ROI (in terms of pixels), and properties within the algorithms itself, whereas for "Live Viewing" (top branch in Fig. 1), the complexity depends linearly on the resolution of the input image sequence. Our approach is to use the previously presented concepts to dynamically switch between quality levels. The dynamics in the latency and throughput can be minimized by incidentally reconfiguring the computing tasks in the flow graph, i.e. freeing or consuming some of the resources that were prior budgeted for background tasks.

Within motion compensated feature enhancement, the latency at the output may vary over time. However, during a live interventional X-ray procedure, large latency differences between succeeding frames are not allowed for clinical reasons (eye-hand coordination of the physician). A straightforward solution is to employ a task partitioning on the platform, based on a worst-case resource reservation. Subsequently, at the end of the pipeline, a task with a variable delay keeps the latency and throughput constant. The main drawback is that for most of the time, the reserved resource budget is set too conservative. Furthermore, the output latency is much higher than actually required and it is impossible to exploit the difference between average case and worst-case requirements without affecting the reliability (guaranteed performance) of the application.

For the (data-dependent) image analysis application, the worst-case execution has heavy excursions (85%) on the effective latency. The computation latency can vary between 60 and 120 ms, when the partitioning of tasks across processing cores is fixed. By exploiting the presented hierarchical QoS concept for latency optimization, the partitioning of tasks across processing cores is dynamically changed when there is a sudden change in resource requirements. As a result, the variation on the latency is reduced significantly to only 20% [see Fig. 11 (top)]. For the static live viewing application running in parallel, the latency varies around 30 ms. This variation results from the switching between different quality modes.

Instead of pure image quality control, the experiments concentrate on keeping the output latency constant by dynamically spitting the essential tasks in sub-tasks and reconfiguring those sub-tasks to processor cores. At the same time, the live viewing complexity and the corresponding image quality is decreased for short periods of time with task skipping of non-essential filtering tasks. The actual selection is based on the resource demand for the image analysis tasks, as estimated by the prediction model in the LQoS controller. Please note that the clinical user is mainly interested in fast results from the image-analysis application, and the concurrently operating live viewing application is only offered as a reference.

At quality level Q2, temporal filtering is skipped, which causes some additional blur in the image, but only when the patient or the patient table is moving. At quality level Q3, also spatial filtering is skipped, which causes some additional noise in the image. Switching frequently between quality modes can have a major impact on the perceived image quality. We therefore restrict the QoS controller to switch very often from a low to a medium or high quality mode. Before a switch to a higher quality mode can occur, the required budget needs to be available for several seconds. This avoids the possible annoying flickering between different quality modes.

In the case study, we process uncompressed image frames (1024 $\times$ 1024 pixels, 30 Hz) on a general-purpose multi-core platform, [34], where the live viewing operates at the full frame rate, and image analysis at half the frame rate. The assignment of processing tasks to processor cores is done manually at design time for each application, as the total amount of application scenarios is limited (three quality levels for each application). At runtime, the QoS control selects based on the prediction of required resources, the optimal quality level for each application, and a corresponding mapping of both applications to the processor cores platform, from a sorted list of solutions.

Tasks are not moved from one core to the other during processing of an image. Before the start of processing a new image, the predictor task estimates the required resources on the platform for fluent operation. The QoS control task then selects, based on the available resources and desired quality setting, the best mapping from a precompiled list of scenarios. The overhead of switching

between different mappings is negligible to the overall task processing as switching only takes place at the image level. Practically, switching is minimized to avoid noticeable flickering in the image. As the task mapping is precompiled, a switch will result in queueing a different set of tasks for the next image from a sorted list of quality levels. The GQoS is storing this sorted list of quality changes, which limits the searching algorithm for finding the new scenario to linear complexity.

Evaluation of QoS with task scalability. The hierarchical QoS framework executes image analysis, enhancement and live viewing tasks. Quality level Q1 means full processing with low resource-usage requirements. We perform task splitting at Q2 and Q3, where for Q2, the analysis tasks are split in two, and for Q3 the analysis tasks are split in four sub-tasks. For live viewing, we defined three quality levels, corresponding to changing the image quality for the application from "highest-quality" to "high-quality" or "basic-quality", which means skipping of parts of spatial or temporal filtering tasks. Let us discuss the dynamic behavior in the scene. We focus on times $t1 \ldots t10$ in Fig. 11 (bottom). At time $t1$, Job 1 changes its requirements on the resources (due to ROI processing) and releases some of its resources to the system. This increases the quality level of Job 2 from Q1 to Q3. At time $t2$, Job 1 changes its requirements on the resources and requests more resources from the system (due to FULL processing). A request for task splitting is initiated by Job 1 and this decreases the quality level of Job 2 back from Q3 to Q1. At time $t3$, Job 1 releases resources and subsequently, Job 2 increases its quality level. The described behavior continues till $t8$, where for both jobs, quality level Q2 is assigned. At $t9$, a resource-usage increase requests task splitting for Job 1, where for Job 2 the quality is decreased. At $t10$, again for both Jobs, quality level Q2 is assigned. The large fluctuations in execution time occur due to image analysis tasks performed in certain regions of the image.

5 Conclusions

In this paper, we have extended QoS control for a set of multiple parallel tasks running on a multi-core processor system. The extension involves the use of image analysis processing, which features a high variation in computing and memory requirements, as apposed to regular stream-oriented video processing. For this extension, we have developed a Markov-based model that captures the computing behavior of the analysis tasks with a reasonable accuracy. This extension has been added to a hierarchical QoS concept that was developed for MPEG-4 coding. We have shown that the overall system is able to predict and handle fluctuations in computing requirements and with task skipping and splitting scalability, the quality is dynamically optimized.

Performance modeling is indispensable for obtaining a high efficiency in the mapping or to increase the functionality of the applications executed on the platform. For our case study involving dynamic image analysis, the timing model for the unpredictable behavior is build with statistical techniques, in particular scenario-based Markov chains. The comparison with the mostly used worst-case approach for resource allocation revealed that it can save an impressive factor of 1.8 on computations for selective cases. The linear models that were found earlier for predicting computations for streaming-based video functions in [28] were equally applicable to model the bandwidth usage.

The results on modeling of computing and bandwidth have been validated by a representative case for professional medical imaging. The accuracy of the modeling is high, between 94 and 97% accuracy, with sporadic excursions of the prediction error up to 20–30% for the statistical techniques. This prediction accuracy is so good that it allows resource prediction at runtime, thereby leading to an actively controlled system management. In such a case, the overhead for resource prediction was estimated to be at an acceptable low level, in the range of 1–5%.

We have adopted the concept of hierarchical QoS management from [31], to control a set of video applications executed on a resource-constrained multi-core processor system. The QoS control is split into two layers: a GQoS for overall system control and a LQoS for individual application control. The LQoS control is based on resource-usage estimation, where the GQoS is based on a heuristic optimization algorithm.

The system is reconfigured when the application considerably changes in performance requirements or additional functionally is needed. This has an impact in two aspects. First, to create more budget for extra functionality or a performance increase, the non-essential tasks may be skipped and essential tasks can be split over the computing engines. Second, when new functionality has to be added, the increased combination of tasks needs to be optimized with respect to the individual task budgets, while the user expects an overall system behavior with high quality. To create the desired scalability in applications, we have used task skipping and splitting as a concept because every video application can be quickly made scalable in this way. Task skipping is only possible with the non-essential tasks, whereas task splitting applies to essential tasks.

Prediction of resource utilization for dynamic applications has led to a significant quality improvement of the complete system when resources are selectively distributed

over the available applications. The jitter on the latency of dynamic image analysis is reduced with almost 70% and a constant throughput is achieved. The proposed QoS mechanism runs fast enough to be executed in real time, because it operates only on sorted lists of benefit functions and related parameters.

The presented results in this paper are rather relevant for the near future, since dynamic video applications like image analysis are increasingly found in both the consumer and professional domain. We are convinced that the proposed solution for modeling medical analysis applications with Markov chains can also be applied to QoS control in other domains, such as consumer multimedia or surveillance. This requires that the algorithm is known to such a degree that tasks can be made scalable and the statistical learning of the Markov model can be carried out properly.

Acknowledgments This work has been performed in the ITEA2 project ip07022 HiPiP. Further thanks go to the anonymous reviewers for their constructive comments.

References

1. Albers, R., Suijs, E., de With P.H.N.: Optimization model for memory bandwidth usage in X-ray image enhancement. In: SPIE Electronic Imaging (2008)
2. Anderson, J., Calandrino, J., Devi, U.: Real-time scheduling on multicore platforms. In: Real-Time and Embedded Technology and Applications Symposium, IEEE, pp. 179–190 (2006)
3. Bismuth, V., Vaillant, R.: Elastic registration for stent enhancement in X-ray image sequences. In: IEEE International Conference on Image Processing, pp. 2400–2403 (2008)
4. Bormans, J., Ngoc, N., Deconinck, G., Lafruit, G.: Terminal QoS: advanced resource management for cost-effective multimedia appliances in dynamic contexts. In: Ambient Intelligence: Impact on Embedded System Design, pp. 183–201. Springer, Berlin (2003)
5. Brandenburg, B., Anderson, J.: Integrating hard/soft real-time tasks and best-effort jobs on multiprocessors. In: 19th Euromicro Conference on Real-Time Systems, IEEE, pp. 61–70 (2007)
6. Bril, R.: Real-time scheduling for media processing using conditionally guaranteed budgets. PhD thesis, Technische Universiteit Eindhoven (2004)
7. Bril, R., Hentschel, C., Steffens, E., Gabrani, M., van Loo, G., Gelissen, J.: Multimedia QoS in consumer terminals. In: IEEE Workshop on Signal Processing Systems, pp. 332–343 (2001)
8. Burchard, L., Altenbernd, P.: Estimating decoding times of MPEG-2 video streams. In: International Conference on Image Processing, 2000, vol. 3 (2000)
9. Florent, R., Nosjean, L., Lelong, P., Rongen, P.: Medical viewing system and method for enhancing structures in noisy images. US 2005/0002546 A1 (2002)
10. Florent, R., Nosjean, L., Lelong, P.: System and method for enhancing an object of interest in noisy medical images. US 2006/0133567 A1 (2004)
11. Florent, R., Nosjean, L., Lelong, P.: Medical viewing system and method for spatially enhancing structures in noisy images. US 2008/7340108 (2008)
12. Fortier, P., Michel, H.: Computer Systems Performance Evaluation and Prediction. Digital Press (2003)
13. Frost, V., Melamed, B.: Traffic modeling for telecommunications networks. IEEE Commun. Mag. **32**(3), 70–81 (1994)
14. Gummaraju, J., Rosenblum, M.: Stream programming on general-purpose processors. In: Proceedings of the 38th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 343–354. IEEE Computer Society, Washington, DC, USA, MICRO 38 (2005)
15. Gupta, N.: A VLSI architecture for image registration in real time. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **15**(9), 981–989 (2007)
16. Jain, R.: The Art of Computer Systems Performance Analysis. Wiley, New York (1991)
17. Jaspers, E., Van Der Tol, E., Eindhoven, C., de With, P.H.N.: System-level analysis for MPEG-4 decoding on a multi-processor architecture. In: 16th International Parallel and Distributed Processing Symposium, p. 310 (2002)
18. Kleinrock, L.: Queueing Systems, Volume I: Theory. Wiley, New York (1975)
19. Lee, E., Messerschmitt, D.: Static scheduling of synchronous data flow programs for digital signal processing. IEEE Trans. Comput. **36**(1), 24–35 (1987)
20. Lee, S.: Developments in X-ray diagnostics. Med. Mundi **50**(1), 40–47 (2005)
21. Martello, S., Toth, P.: Knapsack Problems Algorithms and Computer Implementations. Wiley, Chichester (1990)
22. Meier, B.: Percutaneous coronary intervention: past, present and future. Med. Mundi **50**(1), 26–34 (2006)
23. Meijering, E., Niessen, W., Viegever, M.: Retrospective motion correction in digital subtraction angiography: a review. IEEE Trans. Med. Imaging **18**(1), 2–21 (1999)
24. Mietens, S.: Complexity scalable MPEG encoding. PhD thesis, Technische Universiteit Eindhoven (2004)
25. Moeslund, T., Hilton, A., Krnger, V.: A survey of advances in vision-based human motion capture and analysis. Comput. Vis. Image Underst. **104**(2–3), 90–126 (2006)
26. Moser, M., Jokanovic, D., Shiratori, N.: An algorithm for the multidimensional multiple-choice knapsack problem. IEICE Trans. Fundam. Electr. Commun. Comput. Sci. **80**(3), 582–589 (1997)
27. Oppelt, A.E.: Imaging Systems for Medical Diagnostics. Publicis Corporate Publication (2006)
28. Pastrnak, M., de With, P.H.N.: Data storage exploration and bandwidth analysis for distributed MPEG-4 decoding. In: 8th IEEE International Symposium Consumer Electronics, pp. 206–209 (2004)
29. Pastrnak, M., Poplavko, P., Farin, D.: Data-flow timing models of dynamic multimedia applications for multiprocessor systems. In: System-on-Chip for Real-Time Applications, pp. 206–209. International Workshop, IEEE (2004)
30. Pastrnak, M., De With, P., Van Meerbergen, J.: QoS concept for scalable MPEG-4 video object decoding on multimedia (NoC) chips. In: IEEE Transactions on Consumer Electronics, vol. 52(4), pp. 1418–1426 (2006)
31. Pastrnak, M., de With, P.H.N., van Meerbergen, J.: Realization of QoS management using negotiation algorithms for multiprocessor noc. In: Proceedings of IEEE International Symposium on Circuits and Systems (2006)
32. Poplavko, P., Basten, T., Bekooij, M., van Meerbergen, J., Mesman, B.: Task-level timing models for guaranteed performance in multiprocessor networks-on-chip. In: International Conference on Compilers, Architecture and Synthesis for Embedded Systems, pp. 63–72. ACM (2003)

33. Poplavko, P., Basten, T., van Meerbergen, J.: Execution-time prediction for dynamic streaming applications with task-level parallelism. In: 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools, pp. 228–235. IEEE Computer Society (2007)
34. Radhakrishnan, S., Chinthamani, S., Cheng, K.: The Blackford northbridge chipset for the Intel 5000. Micro. IEEE **27**(2), 22–33 (2007)
35. Reinders, J.: (2007) Intel Threading Building Blocks. O'Reilly Media, Inc
36. Rongen, P., Florent, R., Stegehuis, H.: (2005) Viewing system for control of ptca angiograms, US 2008/0045827 A1
37. Van der Schaar, M., Radha, H.: A hybrid temporal-SNR fine-granular scalability for internet video. IEEE Trans. Circuits Syst. Video Technol. **11**(3), 318–331 (2001)
38. Sriram, S., Bhattacharyya, S.: Embedded Multiprocessors: Scheduling and Synchronization. CRC Press, Boca Raton (2000)
39. Thacker, N., Clark, A., Barron, J., Ross Beveridge, J., Courtney, P., Crum, W., Ramesh, V., Clark, C.: Performance characterization in computer vision: a guide to best practices. Comput. Vis. Image Underst. **109**(3), 305–334 (2008)
40. Theelen, B.: Performance modelling for system-level design. PhD thesis, Technische Universiteit Eindhoven (2004)
41. Turaga, D., Vander Schaar, M., Pesquet-Popescu, B.: Complexity scalable motion compensated wavelet video encoding. IEEE Trans. Circuits Syst. Video Technol. **15**(8), 982–993 (2005)
42. Van Eijndhoven, J., Hoogerbrugge, J., Jayram, M., Stravers, P., Terechko, A.: (2005) Cache-coherent heterogeneous multiprocessing as basis for streaming applications. In: Dynamic and Robust Streaming Between Connected Consumer Electronic Devices, pp. 61–80. Springer, Berlin
43. Ykman-Couvreur, C., Nollet, V., Catthoor, F., Corporaal, H.: (2006) Fast multi-dimension multi-choice knapsack heuristic for MP-SoC run-time management. In: International Symposium on System-on-Chip, pp. 1–4 (2006)

Author Biographies

Rob Albers received his B.Sc. degree in Electrical Engineering from the Fontys Polytechnic College, Eindhoven, the Netherlands, in 2002. Subsequently, he pursued his M.Sc. degree in Electrical Engineering at the Eindhoven University of Technology, the Netherlands, from which he graduated in 2005. During his undergraduate studies, he worked, in 2002, at Siemens-VDO on a simulation system for car navigation performance and in 2005, at Bosch Security on smart search and retrieval for surveillance video databases. From 2005 to 2009, he was, as a Ph.D. candidate, involved in a joint project between the Eindhoven University of Technology and Philips Healthcare about modeling, prediction and control of interventional X-ray imaging systems on multi-core processors. He has published a number of papers in international conferences and cooperated within national and international subsidy projects. Since 2009, he is employed at Philips Healthcare in Best, where he is involved in the research and development of interventional X-ray imaging systems. His interests include image- and video processing, multi-core processor architectures and systems engineering.

Peter H.N. de With , IEEE Fellow, obtained his MSc. in electrical engineering from the Eindhoven University of Technology, and his PhD. from Delft University of Technology, the Netherlands. He joined Philips Research Labs, Eindhoven, in 1984, where he worked on video coding for digital recording. From 1985 to 1993, he was involved in several European projects on SDTV and HDTV recording. In this period, he contributed as a principal coding expert to the DV standardization for digital camcording. Between 1994 and 1997, he was leading the design of advanced programmable video architectures at the same lab. In 1996, he became senior TV systems architect and in 1997, he was appointed as full professor at the University of Mannheim, Germany, at the faculty of Computer Engineering. In Mannheim he was heading the chair on Digital Circuitry and Simulation with emphasis on video systems. Between 2000 and 2007, he was with LogicaCMG in Eindhoven as a principal consultant and also professor at the Eindhoven University of Technology, at the faculty of Electrical Engineering. He is now with CycloMedia Technology, The Netherlands. He has written and coauthored over 200 papers on video coding, architectures and their realization. He is a regular teacher of the Philips Technical Training and for other postacademic courses. In 1995 and 2000, he coauthored papers that received the IEEE CES Transactions Paper Award, and in 2004, the VCIP Best Paper Award. In 1996, he obtained a company Invention Award. Mr. de With is IEEE Fellow, advisor to Philips, scientific advisor of the Dutch Imaging school ASCII, IEEE ISCE and board member of various working groups.